UNIVERSIDAD TECNOLÓGICA DE PEREIRA
UNIVERSITÀ DEGLI STUDI DI SALERNO

MAGISTRALE

Methodology proposal to train a model for the task of automatic driving of vehicles with reinforcement learning using cloud computing.

Santiago Marín Mejía

supervised by
PhD. José Alfredo Jaramillo Villegas
Proff. Roberto Tagliaferri

February, 2020

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Society has changed his perception towards vehicles, from leisure objects to a vital part of everyday life. Besides, vehicles manufacturers are being pushed to produce eco-friendly, safe and able to connect vehicles. Recent new technology advances introduced the possibilities to equip vehicles with sensors, like cameras, radars, etc. Within this conditions, academic research in robotics is going from indoors platforms to full-scale vehicles operating with a high level of autonomy(Thrun, 2012).

In 2010, Google produced its Toyota Prius automated car in the USA, by 2012 they introduced the Lexus RX450h and by 2015 a fully automated vehicle with no steer was presented named "Firefly" (Abduljabbar, Dia, Liyanage, & Bagloee, 2019). They now develop automated vehicles through an independent company named "Waymo", they use several sensors on their vehicles including a 360 degree camera to visualize the context of the world while driving (Waymo, 2018).

Reinforcement learning has been used in a broad variety of robotics related tasks, such as videogames (Mnih et al., 2015), robot locomotion (Endo, Morimoto, Matsubara, Nakanishi, & Cheng, 2008; Kohl & Stone, 2004), and autonomous driving (Shalev-Shwartz, Shammah, & Shashua, 2016; Abbeel, Coates, Quigley, & Ng, 2007). One of the difficulties in functional uses of reinforcement learning is the high-dimensionality of state space just as the enormous activity extend. Building up an ideal strategy over such high-complexity space is tedious. Ongoing work in deep reinforcement learning has gained extraordinary ground in learning in a high dimensional space with

the strength of deep neural networks (Pan, You, Wang, & Lu, 2017).

1.1 Justification

It is not difficult to explain the potential benefits of autonomous vehicle control for both military and commercial applications. The benefits range from finding and helping earthquake victims or on the battlefield, or operating in dangerous environments such as minefields, to exploring planets (Unlimited, 2004). Every year, 1.2 million lives are lost to traffic accidents worldwide, and in the United States the number of tragedies is growing. A common element of these accidents is that 94% involve human error. Driving is not as safe or as easy as it should be, while distractions while driving are on the rise. If we look at the data from Tesla and NHTSA, we can see that in the meantime, Tesla cars with autopilot enabled record one accident for every 2.87 million miles driven and without autopilot one accident for every 1.76 million miles driven. In comparison, the most recent NHTSA data shows that in the United States there is one auto accident every 436,000 miles (*Tesla Vehicle Safety Report*, 2019). Thanks to this huge difference in safety and its potential to radically change mobility and transportation, self-driving has captured the attention of both the research community and businesses (Kendall et al., 2018; Rosenzweig & Bartl, 2015).

As a result, the development of different techniques to train the control agent has increased proportionally, one of the most recent approaches is the use of reinforcing learning algorithms. However, one of the biggest obstacles to progress in this area is the difficulty of training reinforcement learning models in a real environment, and performance in the real world scenario is not as expected due to the large differences between the visual appearance of the virtual simulation and the appearance of the real world driving scene (Pan et al., 2017). To solve this problem, there is software to simulate real-world environments and scaled vehicles to implement these trained algorithms.

1.2 Problem definition

The autonomous driving of vehicles is an issue that has been strongly developed for some years now, can be divided into several levels, from level 0 where there is no help, through level 1 where the help is only in acceleration or braking, to level 5 where the vehicle is in the ability to take full control of driving and the driver becomes a passenger more (NHTSA, 2019). Currently, deep learning algorithms in neural networks are used to make a vehicle reach level 5 with performance similar to that of a human. The training of these algorithms has been done with supervised learning for decades. The problem with this approach is that model performance is highly dependent on the quality and quantity of data, and driving policies are too simplistic to handle complicated real-world cases (Chen, Chen, Zhang, & Hu, 2019; Liang, Wang, Yang, & Xing, n.d.; Xu, Tan, & Kong, 2018). Another technique used is learning by imitation, which aims to learn a driving policy by observing expert demonstrations. However, the lack of expert databases that mimic each potential scenario makes this approach difficult to scale.

In relation to the obstacles presented by the supervised learning algorithms, a system is required that is capable of determining long-term driving strategies, what is called a generalizable and scalable "driving policy". It is beneficial to have a stronger control policy that takes into account a large number of environmental reactions, collisions and off-road conditions for autonomous driving. Deep Reinforcement Learning (RL) offers, a reasonable system for learning such policies from exploration (Večerík et al., 2017) because of its expertise in action planning, such as the Deep Q-Network (DQN) (Mnih et al., 2015) which has also demonstrated the ability to achieve superhuman performance and has had unprecedented success even in the most complex domains such as the game GO and Atari 2600 without any prior human knowledge (Mnih et al., 2016; Silver et al., 2017). However, these algorithms require agents to interact with the environment and unwanted events to occur. Due to the above, training autonomous vehicles using RL in the real world would cause damage to other vehicles and their surroundings so the training of these algorithms in recent studies has been done through simulations.

However, although these agents achieve superhuman performance in simulators when tested in real environments, their performance is poor due to

visual differences, unknown dynamics and uncertainty in system parameters (Pan et al., 2017; Genc et al., 2019). Consequently, an attempt has been made to close this gap between the real and the virtual with the appearance of different simulators, one of which is offered by Amazon DeepRacer, which is a platform that allows us to train reinforced learning algorithms in its simulator and then be deployed in a vehicle to scale and test the robustness of the algorithms.

Thus, it is proposed to train a reinforced learning algorithm in the cloud simulator offered by Amazon SageMaker and Amazon RoboMaker able to complete the tracks available in the Amazon RoboMaker environment.

1.3 General Objective and Specific Objectives

1.3.1 General Objective

Implement an algorithm for a self-powered vehicle in the Amazon Web Services RoboMaker virtual environment using reinforcement learning strategies that give the agent the ability to navigate a track without a definition of pre-established rules.

1.3.2 Specific objectives

1. Evaluate and understand the Amazon SageMaker platform and its different functionalities.

2. Analyze reward functions for reinforcement learning that best suit autonomous driving problems.

3. Implement a metric to evaluate the performance of the different reward functions used in the algorithm.

4. Execute and evaluate the different reward functions and choose the one with the best performance.

Chapter 2

Background

This chapter will describe the theory behind machine learning, reinforcement learning, and cloud computing to understand how they work and how this project is built around these concepts.

2.1 Machine Learning

A machine learning algorithm is an algorithm that can learn from data. Machine learning enables us to tackle tasks that are too difficult to solve with fixed programs written and designed by human beings. From a scientific and philosophical point of view, machine learning is exciting because developing our understanding of it entails developing our understanding of the principles that underlie intelligence.(Goodfellow, Bengio, & Courville, 2016) Usually, machine learning algorithms are divided into two classes supervised learning and unsupervised learning.

In supervised learning, the algorithm is given a pair of inputs (x, y) related to an unknown function $y = f(x)$, the algorithm then tries to learn f estimating $p(y|x)$, if successful, it will be able to calculate any (y) given a new (x). For example, in the task of image recognition, the algorithm would be given an image (x) with a label (y), like cars and bikes, after training the algorithm for some time it would be able to differentiate image containing cars from images containing bikes, and it will be able to classify new images into one of both categories.

In unsupervised learning, only (x) is given to the system for the training, and then the algorithm is expected to find patterns and group the data into k number of groups that share similar characteristics.

There are other kinds of machine learning algorithms like the reinforcement learning algorithm, which is based on a reward function and a set of actions, it tries to maximize the reward chosen the best action based on its interaction with the environment and its actual situation.

2.2 Reinforcement Learning

Reinforcement learning is learning what to do—how to map situations to actions—so as to maximize a numerical reward signal. The agent has to discover which set of actions depending on the situation will return a higher reward, this trough trial and error, it has to take into account not only the actual reward but also the possible reward in the future (Sutton & Barto, 2017)

The agent in a given environment ε and an initial state S is given a set of actions called the action space denoted as A. On every time t this agent in a state s_t will proceed to receive an Observation o_t with a Reward r_t, then it will choose an Action a_t from A moving the agent to a new state s_{t+1} and receive a new observation o_{t+1} with a reward r_{t+1}. The goal of the agent is to choose a_t from A aiming to receive the highest r_t possible, the selection of actions can be made based on previous iterations or even randomly.

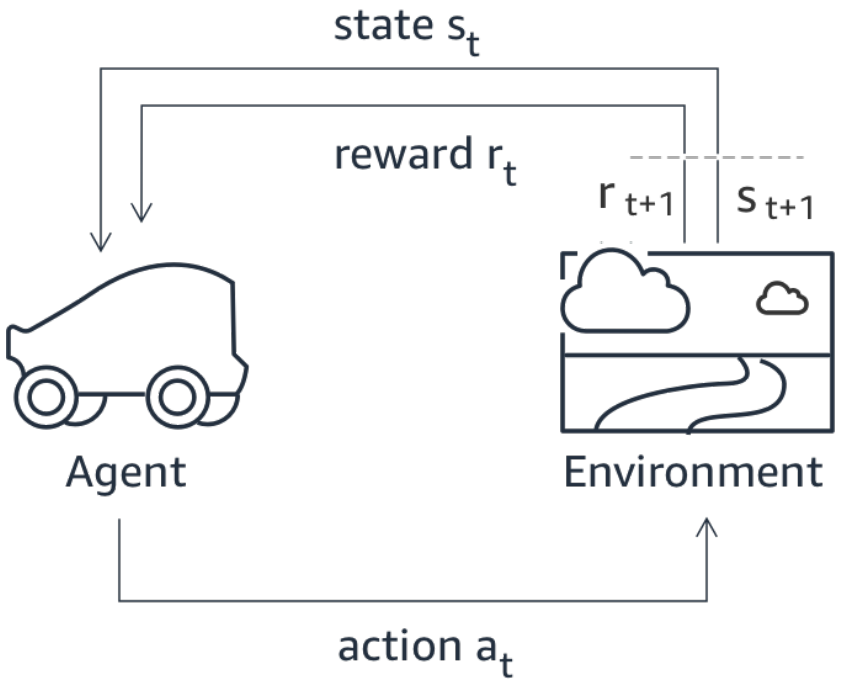

Figure 2.1: Reinforcement learning overview

Since the agent has access to partial information about the task from its actual state in the current screen, it is impossible to completely understand the situation from x_t. As a result, It's considered sequences of actions and observations, $s_t = x_1$, a_1, x_2, ..., a_{t-1}, x_t, and learn strategies that depend upon these sequences. All sequences are assumed to terminate in a finite number of time-steps.This arises a large but finite Markov decision process (MDP) where each sequence is a separate state. Therefore, conventional reinforcement learning methods for MDPs can be applied by using all the sequence s_t as the state at time t(Mnih & Silver, 2013).

Exploration and Exploitation

Every reinforcement learning algorithm needs to balance between exploration and exploitation. The algorithm has to explore different unexplored actions that may not return any rewards. The agent should explore all the states and actions in the action space to discover which actions return higher rewards. The agent should also start using those actions that mean higher reward so that the model converges to a solution. This is known as the exploration and exploitation trade-off. It decrease the plausibility that the agent might learn a less efficient policy. (Sadeghi, Toshev, & Jang, 2017)

2.3 Cloud Computing

Cloud computing is a model that enables access to a shared pool of flexible and configurable computing resources (servers, storage, networks, services, and applications), can be managed and provisiones with minimum management effort. The model is constituted of five essential characteristics, three service models, and four deployment models.(Mell & Grance, 2011)

Essential Characteristics:

On-demand self-service. The user can provision server time and storage, according to necessities without requiring direct interaction with each service.

Broad network access. Services are available to be accessed from a wide variety of platforms (e.g., mobile phones, tablets, laptops, and workstations).

Resource pooling. The provider serves multiple consumers using a multi-tenant model, customers, or "tenants" with provisional and scalable services. These services are flexible and can be adjusted depending on the need of the users without affecting any of them. Examples of resources include storage, processing, memory, and network bandwidth.

Rapid elasticity. The services provided are rapidly scalable and answer automatically in many cases, for the user it could seem like the resources are unlimited, and they are always ready following the demand.

Measured service. To be able always to fulfill the client demand with the flexibility of the service and the scalability of the services (e.g., storage, processing, bandwidth, and active user accounts). the cloud system has to monitor the resources, providing transparency for both the provider and consumer of the utilized service.

Service Models:

Software as a Service(SaaS). It grants the user the capability to run the provider's applications on the cloud infraestructure. The applications can be accessed from different devices, such a program interface or a web browser.

The underlying cloud infraestructure like servers, network, storage, and operating systems are not managed by the client but by the cloud provider.

Platform as a Service (PaaS). It gives the user the possibility to deploy on the cloud infraestructure the applications aquired or created using libraries, programming language, tools, and libraries supported by the cloud provider. The cloud infraestructure like servers, operating systems, storage or networks are nor controlled by the customer, but has control over the configuration settings of the environment and the applications.

Infrastructure as a Service(IaaS). It presents the user the capability to provision storage, networks, processing, among other computing resources in which the customer can deploy and run any software, including applications and operating systems. The cloud infraestructure is not controlled by the user but he can control over operating systems, storage and configuration settings of the environment and the applications.

Deployment Models:

Private cloud. The cloud infrastructure is provisioned for private use by a single organization, including various users. It may be owned, controlled, and managed by the company, a third party, or the combination of them.

Community cloud. The cloud infrastructure is provisioned for private use by a particular association of users from groups that have shared interests(e.g., mission, security requirements, policy,and compliance considerations). It may be maintained, controlled, and managed by one or more of the groups in the association, a third party, or a combination of them.

Public cloud. The cloud infrastructure is provisioned for public use. It may be maintained, controlled, and managed by a company, government, or academic organization, or some combination of them.

Hybrid cloud. The cloud infrastructure is a combination of two or more cloud infraestructures (public, private, or community) by allowing information flow between public and private clouds. It gives companies greater flexiblity and allows application portability.

2.4 Amazon Web Services

To train the agent in this project Amazon Web Services (AWS) was used. AWS offers instances that can be scaled depending on the needs and budget of the user. It offers over 175 services, among which are Amazon SageMaker, Amazon RoboMaker, and Amazon DeepRacer. In this section will be discussed the most relevant aspects of each one.

2.4.1 AWS SageMaker

AWS SageMaker is a web service that provides the user the opportunity to build, train and deploy machine learning models, all of these using Amazon cloud computing services where the customers can choose from different configurations according to their needs and budget.

2.4.2 AWS RoboMaker

With AWS RoboMaker the customer is able to run simulation where robots can stream data, navigate, communicate, comprehend and learn. RoboMaker provides AWS DeepRacer the perfect environment to train and evaluate the agents. For the simulation engine it uses open source Gazebo engine. The default physics engine is ODE (Open Dynamics Engine). The default rendering engine is OGRE (Object-Oriented Graphics Rendering Engine).

Figure 2.2: RoboMaker simulation environment

2.4.3 AWS DeepRacer

Amazon DeepRacer is a new AWS service that provides developers of any skill level to get started with machine learning with hands-on tutorials and guidance on building reinforcement learning models. It has an integrated simulation environment hosted on the AWS Cloud for experimentation and optimization of your autonomous racing models, built with reinforcement learning.

In AWS DeepRacer the developers can choose the action space they want to use, varying the steering angles and the speed that the agent can take; besides which track they want to use in the simulation for the training, but the most relevant part is that developers can build their reward function. The reward function is a Python function which is given specific parameters that describe the current state and returns a numeric reward value.

The parameters passed to the reward function describe various aspects of the state of the vehicle, such as its position and orientation on the track, its observed speed, steering angle and more.

We will explore some of these parameters and how they describe the vehicle as it drives around the track:

- Position on track
- Heading
- Waypoints
- Track width
- Distance from center line
- All wheels on track
- Speed
- Steering angle

AWS DeepRacer integrates with Amazon SageMaker for reinforcement learning model training, AWS RoboMaker to provide the racing simulator, Amazon Kinesis Video Streams for video streaming of virtual simulation footage, Amazon S3 for model storage, and Amazon CloudWatch for log capture.

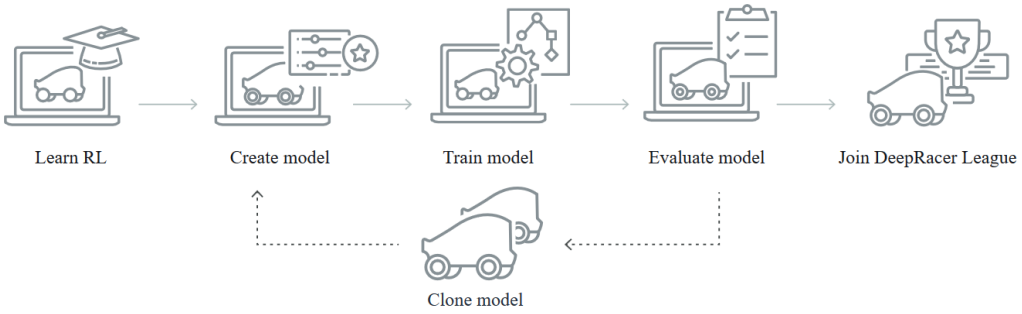

Figure 2.3: AWS DeepRacer overview

Chapter 3

Method

In this chapter the methodology used on the experiments will be explained.

3.1 Equipment

The training of the network was made on an Amazon SageMaker ml.t2.medium instance equipped with 2 virtual CPUs and 4 GB of memory. The simulated track to train the agent was made in the Amazon RoboMaker service.

3.2 Model

The model used on the Proximal Policy Optimization (PPO), a state-of-the-art policy gradient algorithm(Sadeghi et al., 2017). The algorithm used two neural network during the training, one is a policy network and a value network. The policy network chooses an action based on the input image from the agent and the value network estimates the discounted reward given the actual state. The policy network collect data. The resulting dataset is then used to update the policy and value networks. The updated policy then interacts with the environment and collects new data and the training continues until a time limit.

3.3 Action Space

The action space used in the experiments had two parameters on each one, the speed and the steering angle, the speed in the set of actions ranged from $0,33\ m/s$ to $1\ m/s$ and the steering angle ranged from -30° to 30°, the next table shows the action space used:

Table 3.1: Action space with 15 actions

Action number	**Steering**	**Speed**
1	-30 degrees	0,33 m/s
2	-30 degrees	0,66 m/s
3	-30 degrees	1 m/s
4	-15 degrees	0,33 m/s
5	-15 degrees	0,66 m/s
6	-15 degrees	1 m/s
7	0 degrees	0,33 m/s
8	0 degrees	0,66 m/s
9	0 degrees	1 m/s
10	15 degrees	0,33 m/s
11	15 degrees	0,66 m/s
12	15 degrees	1 m/s
13	30 degrees	0,33 m/s
14	30 degrees	0,66 m/s
15	30 degrees	1 m/s

3.4 Reward Function

When training and agent using reinforcement learning one of the essential parts is the reward function, for this project three reward functions were tested, the first one is the default reward function offered by AWS DeepRacer, this reward function gives a higher reward to the agent for staying inside the track, no matter where as long as it stays inside the track; the second reward function also reward higher speed and gives more reward depending on the progress of the agent; and the third reward function, takes into account the

speed, the progress, and also the direction which the agent is heading, this to prevent zigzagging.

3.4.1 First reward function "Stay inside the two borders"

This reward function gives a higher reward to the agent for staying inside the border lines of the road, no matter where as long as it stays inside the track.

```
def reward_function(params):
    '''
    Example of rewarding the agent to stay inside the two borders of the track
    '''

    # Read input parameters
    all_wheels_on_track = params['all_wheels_on_track']
    distance_from_center = params['distance_from_center']
    track_width = params['track_width']

    # Give a very low reward by default
    reward = 1e-3

    # Give a high reward if no wheels go off the track and
    # the agent is somewhere in between the track borders
    if all_wheels_on_track and \
    (0.5*track_width - distance_from_center) >= 0.05:
        reward = 1.0

    # Always return a float value
    return float(reward)
```

3.4.2 Second reward function "Reward speed and progress"

To give a reward for being inside the two borders of the track can be a very simplistic reward, it may be useful for generalizing but not so good for efficiency, this reward is not motivating the agent to go faster, nor to be more

efficient. This new reward function gives extra reward if the agent goes faster and also gives an extra reward depending on the progress of the agent in the track, so the further it goes, the higher the reward will be.

```
def reward_function(params):
    '''
    Example of rewarding the agent to stay inside the two borders of the track
    '''

    # Read input parameters
    all_wheels_on_track = params['all_wheels_on_track']
    distance_from_center = params['distance_from_center']
    track_width = params['track_width']
    speed = params['speed']
    progress = params['progress']

    # Set the speed threshold based your action space
    SPEED_THRESHOLD = 0.67
    # Give a very low reward by default
    reward = 1e-3

    # Give a high reward if no wheels go off the track and
    # the agent is somewhere in between the track borders
    if all_wheels_on_track and \
    (0.5*track_width - distance_from_center) >= 0.05 and \
    speed>=SPEED_THRESHOLD:
        reward = 1.0+(progress/100)
    elif all_wheels_on_track and \
    (0.5*track_width - distance_from_center) >= 0.05 and \
    speed < SPEED_THRESHOLD:
        # Penalize if the car goes too slow
        reward = 0.5+(progress/100)

    # Always return a float value
    return float(reward)
```

3.4.3 Third reward function "Reward speed, progress and direction"

The last reward function tries to tackle another obstacle observed with the last two reward functions, the zigzagging, to do this, this reward function gives an extra reward every time the agent is facing no more than 10 degrees away from the next waypoint in the track.

```
def reward_function(params):
    '''
    Example of rewarding the agent to stay inside the two borders of the track
    '''
    import math

    # Read input variables
    waypoints = params['waypoints']
    closest_waypoints = params['closest_waypoints']
    heading = params['heading']
    all_wheels_on_track = params['all_wheels_on_track']
    distance_from_center = params['distance_from_center']
    track_width = params['track_width']
    speed = params['speed']
    progress = params['progress']

    # Calculate the direction of the center line based on the closest waypoints
    next_point = waypoints[closest_waypoints[1]]
    prev_point = waypoints[closest_waypoints[0]]
    # Calculate the direction in radius, arctan2(dy, dx), the result is
    #(-pi, pi) in radians
    track_direction = math.atan2(next_point[1] - prev_point[1], \
                                 next_point[0] - prev_point[0])
    # Convert to degree
    track_direction = math.degrees(track_direction)

    # Set the speed threshold based your action space
    SPEED_THRESHOLD = 0.67
    # Give a very low reward by default
    reward = 1e-3
```

```
# Give a high reward if no wheels go off the track and
# the agent is somewhere in between the track borders
if all_wheels_on_track and \
(0.5*track_width - distance_from_center) >= 0.05 and \
speed>=SPEED_THRESHOLD:
    reward = 1.0+(progress/100)
elif all_wheels_on_track and \
(0.5*track_width - distance_from_center) >= 0.05 and \
speed < SPEED_THRESHOLD:
    # Penalize if the car goes too slow
    reward = 0.5+(progress/100)
# Calculate the difference between the track direction and the heading
#direction of the car
direction_diff = abs(track_direction - heading)
if direction_diff > 180:
    direction_diff = 360 - direction_diff

# Penalize the reward if the difference is too large
DIRECTION_THRESHOLD = 10.0
if direction_diff > DIRECTION_THRESHOLD:
    reward *= 0.5

# Always return a float value
return float(reward)
```

3.5 Training

To better understand the way the training conditions can affect the results of a model, the reward functions were trained in two different instances, first on an easy track, and a hard track. The agents were trained in the AWS DeepRacer server, as mentioned in section 3.1, different agents were used for each reward function, and the training was done for one hour with the model described in section 3.2.
For the first training, the re:Invent 2019 track was used for the training; this track was the official 2019 DeepRacer League Summit Circuit track, it has a length of 17.6 m, a width of 76 cm and is considered an easy track.

For the second training, the Cumulo Carrera track was used, this was the training track for the Virtual Circuit World Tour in September 2019. It has a length of 20.51 m, a width of 76 cm, and is considered a hard track.

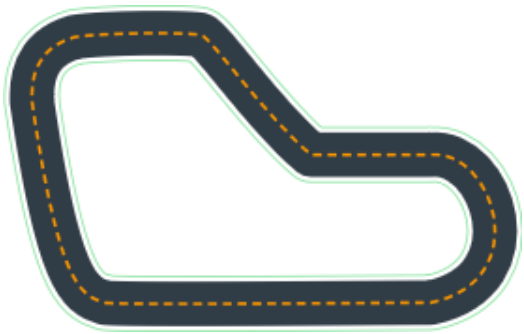

Figure 3.1: re:Invent 2018 track

For the second training, the Cumulo Carrera track was used, this was the training track for the Virtual Circuit World Tour in September 2019. It has a length of 20.51 m, a width of 76 cm, and is considered a hard track.

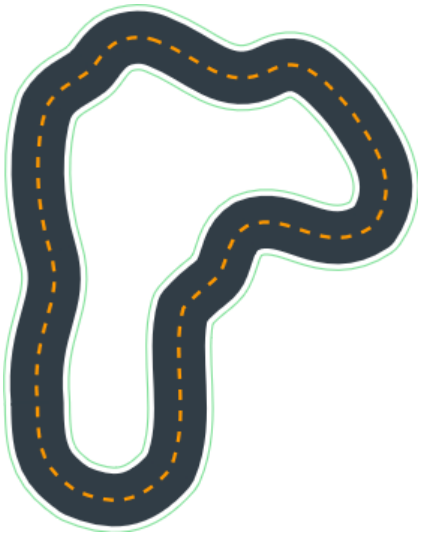

Figure 3.2: Cumulo Carrera track

3.6 Evaluation

The evaluation was done in four different tracks; the tests were done five times on each track, controlling the completion percentage and the time on each track, the tracks used for the evaluation are shown next:

re:Invent 2018 track

The official 2019 DeepRacer League Summit Circuit track.

- Length: 17.6 m (57.97')
- Width: 76 cm (30")
- Difficulty: Easy

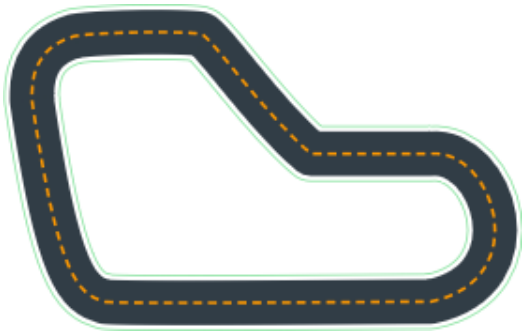

Figure 3.3: re:Invent 2018 track

Bowtie track

The Bowtie offers a simple symmetrical track with a twist. The track features shallow turns that you can use to experiment with different racing behaviors.

- Length: 17.43 m (57.19')
- Width: 76 cm (30")
- Difficulty: Easy

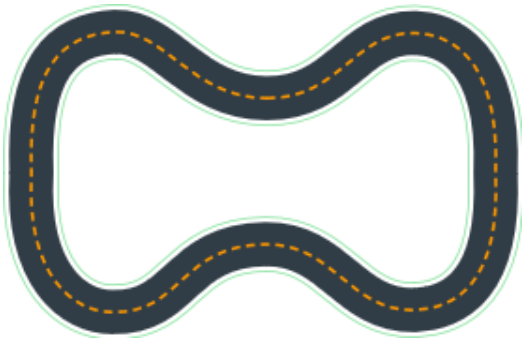

Figure 3.4: Bowtie track

London Loop track

This is the training track for the Virtual Circuit World Tour in May 2019.

- Length: 19.45 m (63.81')
- Width: 76 cm (30")
- Difficulty: Medium

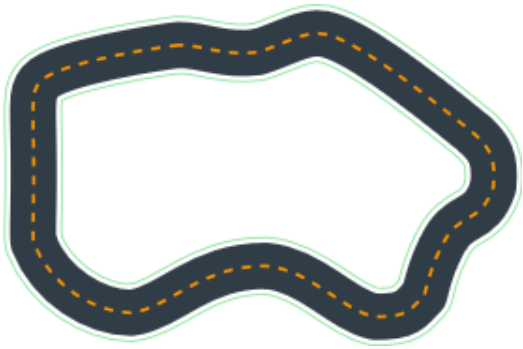

Figure 3.5: London Loop track

Cumulo Carrera track

This is the training track for the Virtual Circuit World Tour in September 2019.

- Length: 20.51 m (67.29')

- Width: 76 cm (30”)
- Difficulty: Hard

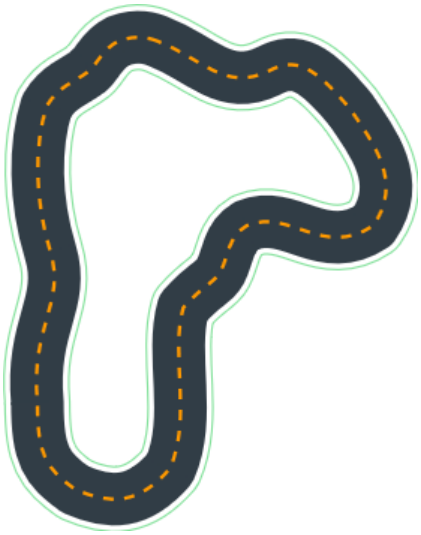

Figure 3.6: Cumulo Carrera track

3.6.1 Overall Performance Metric

To compare and measure the overall performance of each reward function we created a metric to take into account the percentage of completion, the time taken to complete the track and the times the reward function was able to complete the track. The function created for the metric is:

$$\frac{CP * (TC + 1)}{t} \tag{3.1}$$

Being CP the Completion Percentage, TC the Times Completed the tracks and t the time taken to complete the track. This metric takes into account the average completion percentage of all the evaluation in all the tracks, the average time taken to complete the track, and the times that the agent could complete the track (100%). The higher the value on the metric means a better overall performance of the reward function on the evaluations.

Chapter 4

Results

The following chapter will present the results divided into two sections, section 4.1 will show the results from the different reward functions trained on the easy track, and section 4.2 will present the results from the reward functions trained on the hard track.
The results will be presented on tables showing the five trial, their respective completion percentage and the time that the agent took, furthermore, at the end of each table and average of completion and time will be shown, it is important to note that the average time will only take into account the trials where the completion percentage was 100%.

4.1 Training on the easy track

The first approach to the training problem was to use an easy track to train the agent, the re:Invent track, shown on figure 3.6, was the track used for this training task, now we will present the results for the training for each reward function.
This section is divided into two subsections, subsection 4.1.1 will present the graphics from the training, showing with a green line the reward from every episode and with a purple line the track completion in percentage; and subsection 4.2.2 will show the evaluation results from every reward function on each track.

4.1.1 Training

First reward function "Stay inside the two borders"

Here the training and evaluation results for the reward function presented on subsection 3.4.1 will be presented:

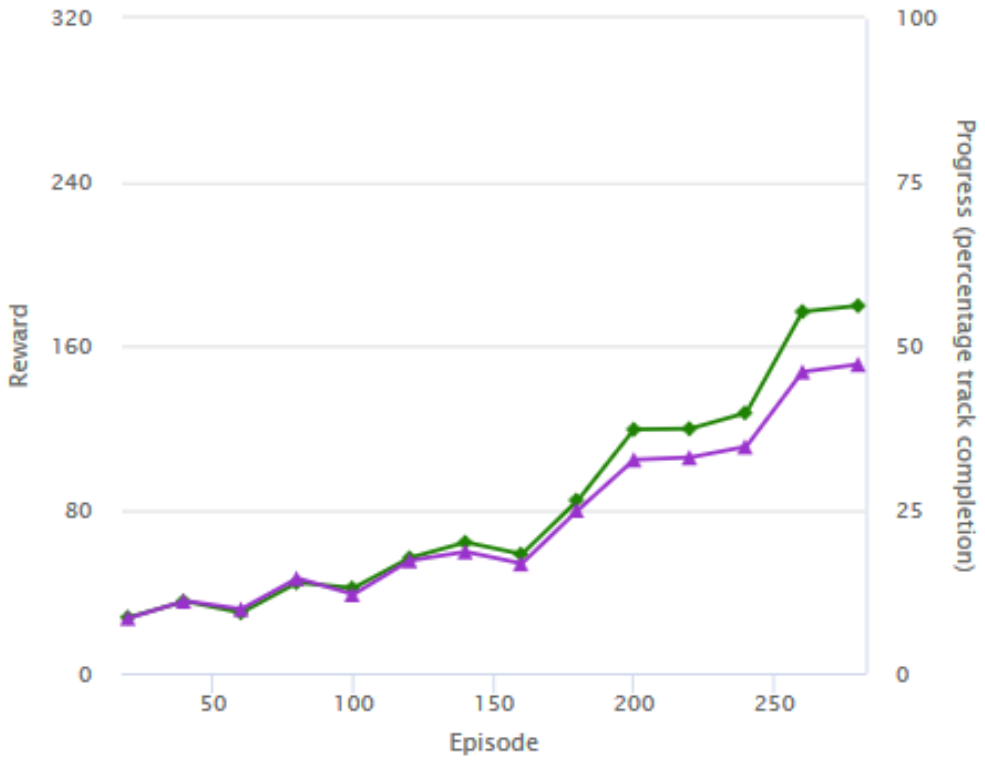

Figure 4.1: First reward function training graphic on the easy track

Second reward function "Reward speed and progress"

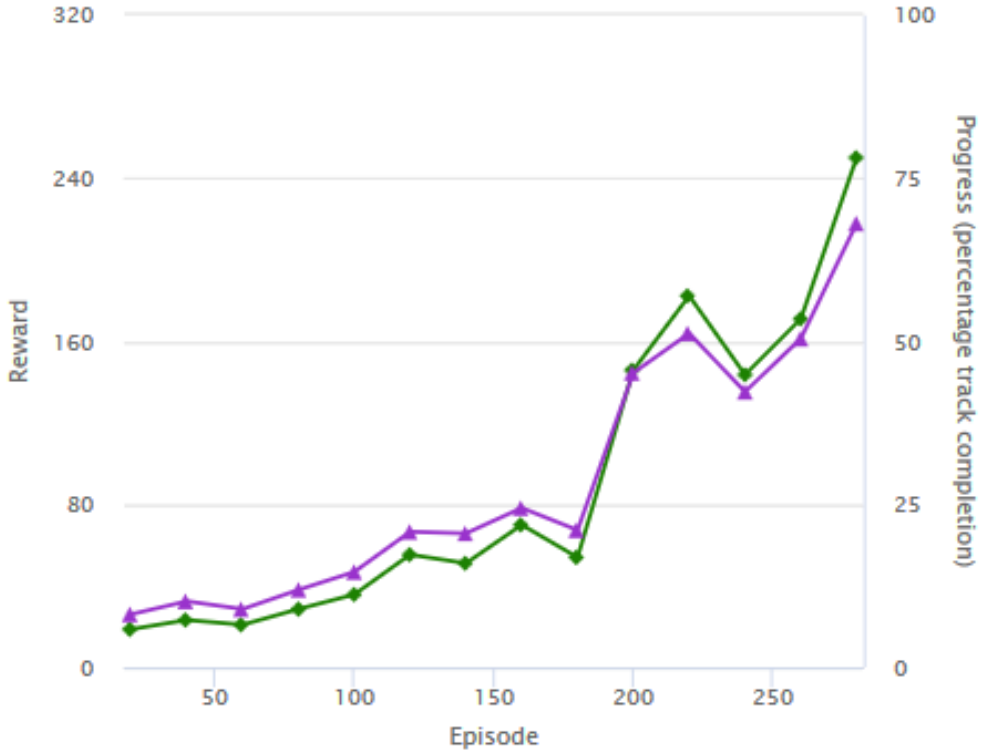

Figure 4.2: Second reward function training graphic on the easy track

Third reward function "Reward speed, progress and direction"

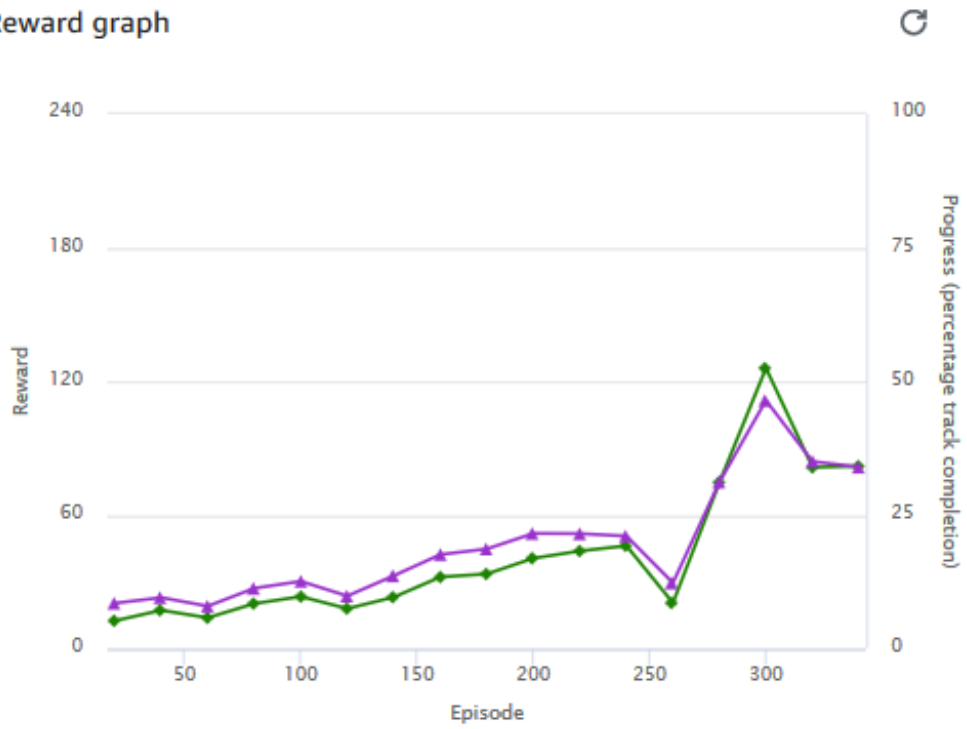

Figure 4.3: Third reward function training graphic on the easy track

4.1.2 Evaluation

For this evaluation, four different tracks were used, the tracks used were described in section 3.6. Also, we will explain the metric used to measure how good the reward functions are.

First reward function "Stay inside the two borders"

Trial	Track completion %	Time
1	100	28,303
2	30	7,805
3	100	27,183
4	100	27,183
5	100	29,01
Average	86	27,91975

Table 4.1: Evaluation in re:Invent 2018 track

Trial	Track completion %	Time
1	100	28,382
2	39	12,903
3	48	13,904
4	100	28,32
5	100	30,523
Average	77,4	29,075

Table 4.2: Evaluation in Bowtie track

Trial	Track completion %	Time
1	19	7,029
2	78	25,797
3	25	8,965
4	9	2,95
5	11	3,829
Average	28,4	0

Table 4.3: Evaluation in London Loop Track

Trial	Track completion %	Time
1	51	19,592
2	58	22,131
3	55	21,894
4	33	12,916
5	60	22,557
Average	51,4	0

Table 4.4: Evaluation in Cumulo Carrera Track

The results from table 4.1 show that this reward function has an excellent performance when it is tested on the same track where it was trained. When tested on another easy track the results were good too as shown on table 4.2, finishing the track on three of five trials, but performing poorly in the medium and hard tracks as shown on tables 4.3 and 4.4 respectively, this can be an evidence that this reward function has problems to generalize to perform in more exigent scenarios.

Second reward function "Reward speed and progress"

Trial	Track completion %	Time
1	100	28,382
2	39	12,903
3	48	13,904
4	100	28,32
5	100	30,523
Average	77,4	29,075

Table 4.5: Evaluation in re:Invent 2018 track

Trial	Track completion %	Time
1	49	12,624
2	100	26,704
3	82	23,003
4	100	27,905
5	54	12,264
Average	77	27,3045

Table 4.6: Evaluation in Bowtie track

Trial	Track completion %	Time
1	100	30,559
2	100	32,914
3	18	5,751
4	100	31,514
5	100	29,944
Average	83,6	31,23275

Table 4.7: Evaluation in London Loop Track

Trial	Track completion %	Time
1	89	27,468
2	56	18,556
3	93	27,226
4	26	10,436
5	50	14,938
Average	62,8	0

Table 4.8: Evaluation in Cumulo Carrera Track

According to table 4.5, 4.6 and 4.7, The results for the second reward function shows a better performance when generalizing, also is the reward function with the best performance when it comes to the evaluation of the hard track (Table 4.8), having an average of 62,8%, compared to the 51,4% and the 32% completion percentage accomplished by the first and third reward functions respectively. It is also important to note that even though it rewards high speeds, it got the worst average time on the first track, but a better time on the bowtie track, this could be due to zigzagging.

Third reward function "Reward speed, progress and direction"

Trial	Track completion %	Time
1	49	12,55
2	73	18,76
3	100	24,781
4	100	27,095
5	100	27,931
Average	84,4	26,6023333

Table 4.9: Evaluation in re:Invent 2018 track

Trial	Track completion %	Time
1	28	8,031
2	76	22,741
3	7	2,044
4	51	13,699
5	30	7,492
Average	38,4	0

Table 4.10: Evaluation in Bowtie track

Trial	Track completion %	Time
1	80	23,948
2	59	17,553
3	35	9,721
4	88	27,566
5	24	6,81
Average	57,2	0

Table 4.11: Evaluation in London Loop Track

Trial	Track completion %	Time
1	10	4,168
2	35	10,287
3	51	15,873
4	20	6,513
5	44	12,503
Average	32	0

Table 4.12: Evaluation in Cumulo Carrera Track

This reward function only performed well on the first track where it was trained, it got the best time average on that track but was the worst one when it came to generalizing the driving policy to complete the other tracks, the agent trained under this reward function was not able to complete any trial in any other track on the evaluation phase.

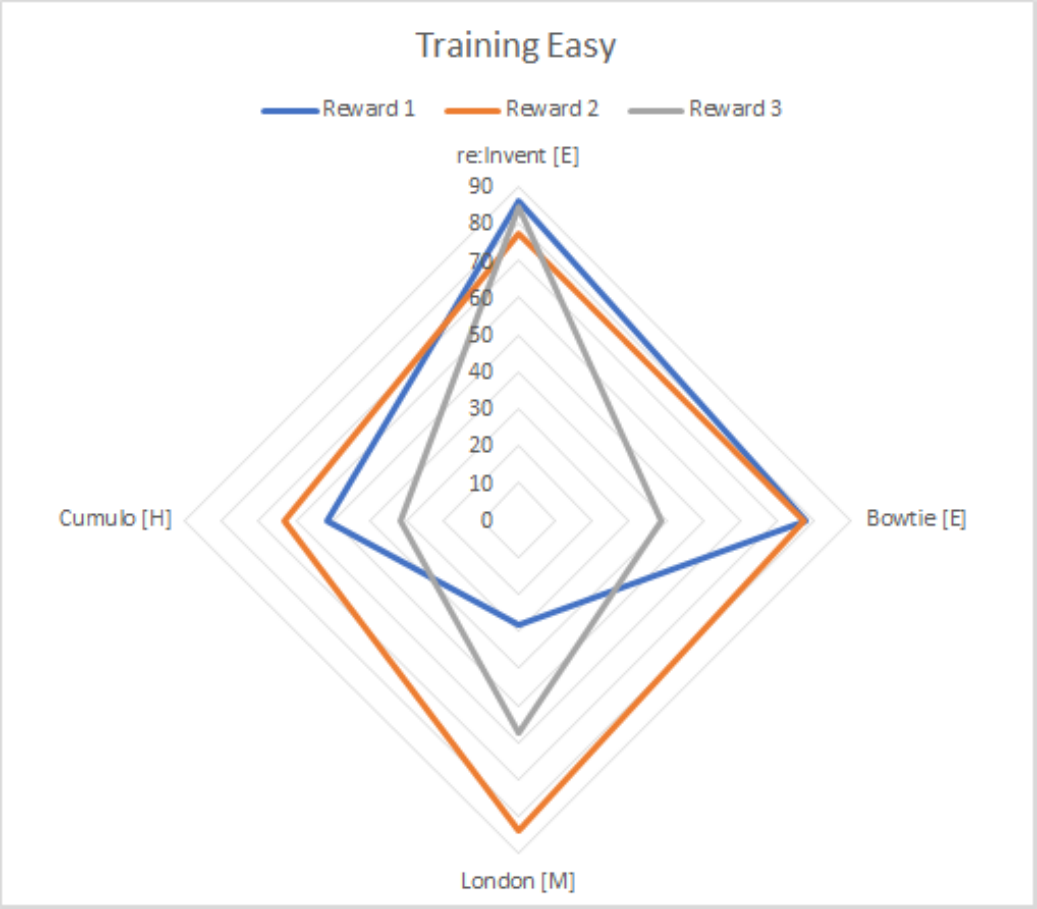

Figure 4.4: Reward functions trained in the easy track evaluation performance overview

Reward Function	CP	t Avg.	TC	Metric
1	60,8	22,5	7	21,61
2	75,2	25,13	9	29,93
3	53	17,76	3	11,93

Table 4.13: Evaluation Summary trained in the Easy track

The Figure 4.4 shows how the second reward function is the one with the best average track completion when evaluated in all the tracks, also, when we

look at the Table 4.13 and compare the overall performance metric we can clearly see how the second reward function has the best performance there too.

4.2 Training on the hard track

The second approach to the training problem was to use a hard track to train the agent, the Cumulo Carrera track, shown on figure 3.9, was the track used for this training task, now we will present the results for the training for each reward function.
This section is divided into two subsections, subsection 4.2.1 will present the graphics from the training, showing with a green line the reward from every episode and with a purple line the track completion in percentage; and subsection 4.2.2 will show the evaluation results from every reward function on each track.

4.2.1 Training

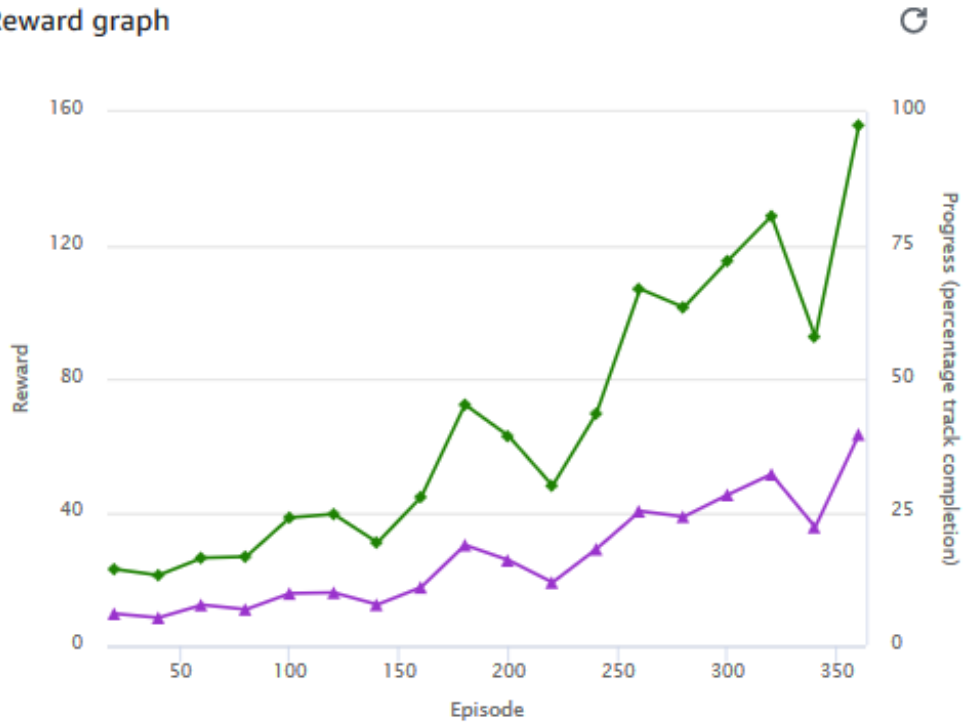

Figure 4.5: First reward function training graphic on the hard track

Second reward function "Reward speed and progress"

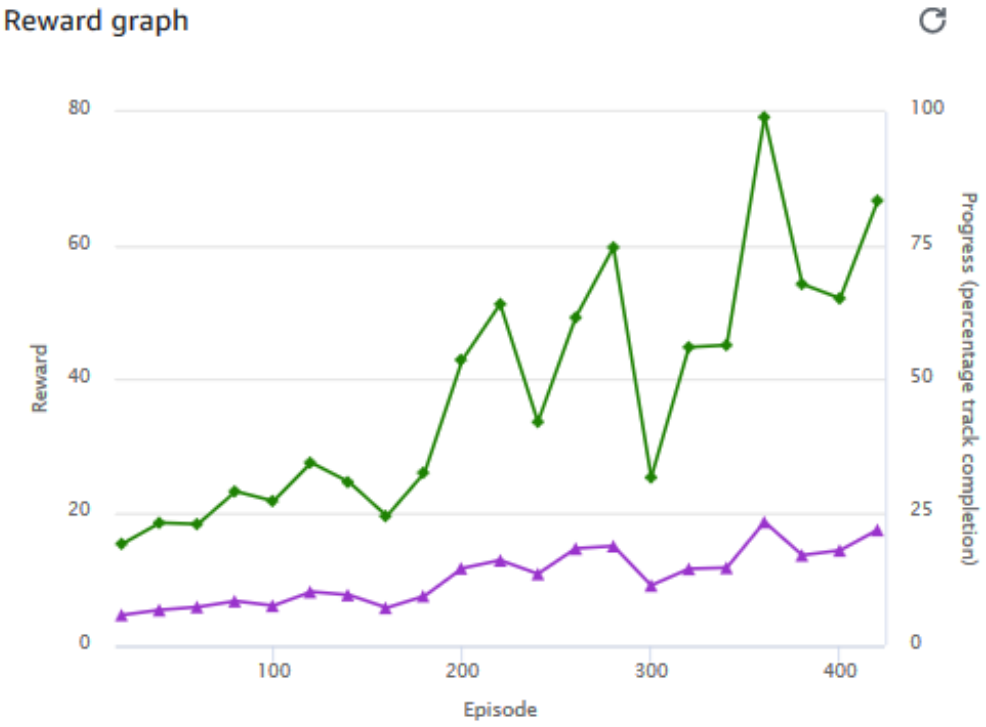

Figure 4.6: Second reward function training graphic on the hard track

Third reward function "Reward speed, progress and direction"

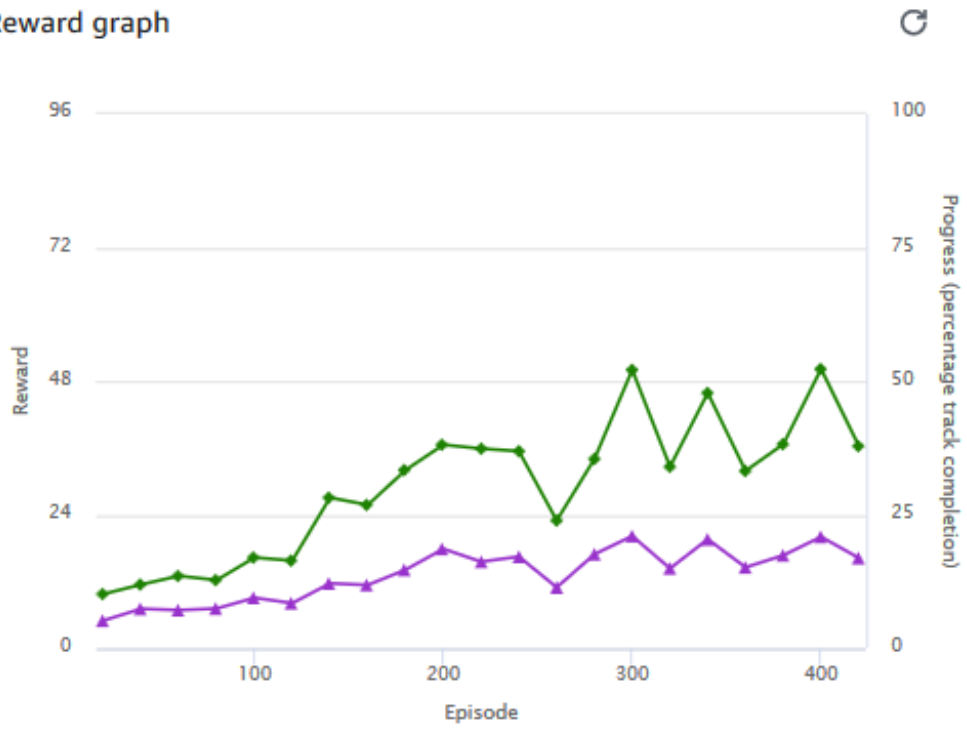

Figure 4.7: Third reward function training graphic on the hard track

4.2.2 Evaluation

For this evaluation, four different tracks were used, the tracks used were described in section 3.6.

First reward function "Stay inside the two borders"

Trial	Track completion %	Time
1	6	2,581
2	56	16,932
3	54	17,464
4	46	15,317
5	6	2,224
Average	26,8	0

Table 4.14: Evaluation in re:Invent 2018 track

Trial	Track completion %	Time
1	29	8,99
2	8	3,069
3	79	24,778
4	8	2,971
5	100	29,979
Average	44,8	29,979

Table 4.15: Evaluation in Bowtie track

Trial	Track completion %	Time
1	100	34,122
2	100	34,464
3	81	26,796
4	100	36,153
5	100	31,889
Average	96,2	34,157

Table 4.16: Evaluation in London Loop Track

Trial	Track completion %	Time
1	43	16,469
2	43	15,926
3	43	16,482
4	29	9,848
5	43	15,179
Average	40,2	0

Table 4.17: Evaluation in Cumulo Carrera Track

This reward function shows an excellent performance in the London Loop track, as shown in table 4.15 but poor performance in the Cumulo Carrera track where it was trained; it also performed poorly in the easy tracks, showing poor generalizing potential again.

Second reward function "Reward speed and progress"

Trial	Track completion %	Time
1	31	9,802
2	40	10,206
3	27	8,24
4	29	7,895
5	7	1,883
Average	38	0

Table 4.18: Evaluation in re:Invent 2018 track

Trial	Track completion %	Time
1	77	18,129
2	18	5,297
3	24	6,126
4	100	22,309
5	100	24,63
Average	63,8	23,4695

Table 4.19: Evaluation in Bowtie track

Trial	Track completion %	Time
1	31	9,376
2	100	28,838
3	100	28,337
4	41	11,02
5	100	28,927
Average	74,4	28,7006667

Table 4.20: Evaluation in London Loop Track

Trial	Track completion %	Time
1	100	30,127
2	100	29,102
3	51	15,535
4	30	8,636
5	43	13,05
Average	64,8	29,6145

Table 4.21: Evaluation in Cumulo Carrera Track

The agent trained with the second reward function displays a better performance on all the tracks except the first one shown in table 4.17, not only could generalize better but also got the best times.

Third reward function "Reward speed, progress and direction"

Trial	Track completion %	Time
1	27	7,901
2	27	7,578
3	83	24,241
4	27	6,445
5	26	8,633
Average	38	0

Table 4.22: Evaluation in re:Invent 2018 track

Trial	Track completion %	Time
1	96	26,965
2	27	10,292
3	79	22,132
4	51	14,262
5	26	9,567
Average	55,8	0

Table 4.23: Evaluation in Bowtie track

Trial	Track completion %	Time
1	100	31,424
2	13	4,122
3	78	22,569
4	100	33,641
5	100	33,822
Average	78,2	32,9623333

Table 4.24: Evaluation in London Loop Track

Trial	Track completion %	Time
1	100	29,749
2	79	24,045
3	28	8,778
4	79	25,943
5	100	31,668
Average	77,2	30,7085

Table 4.25: Evaluation in Cumulo Carrera Track

The agent trained with the third reward function could only perform well on the last two tracks showing again a lack of generalizing potential.

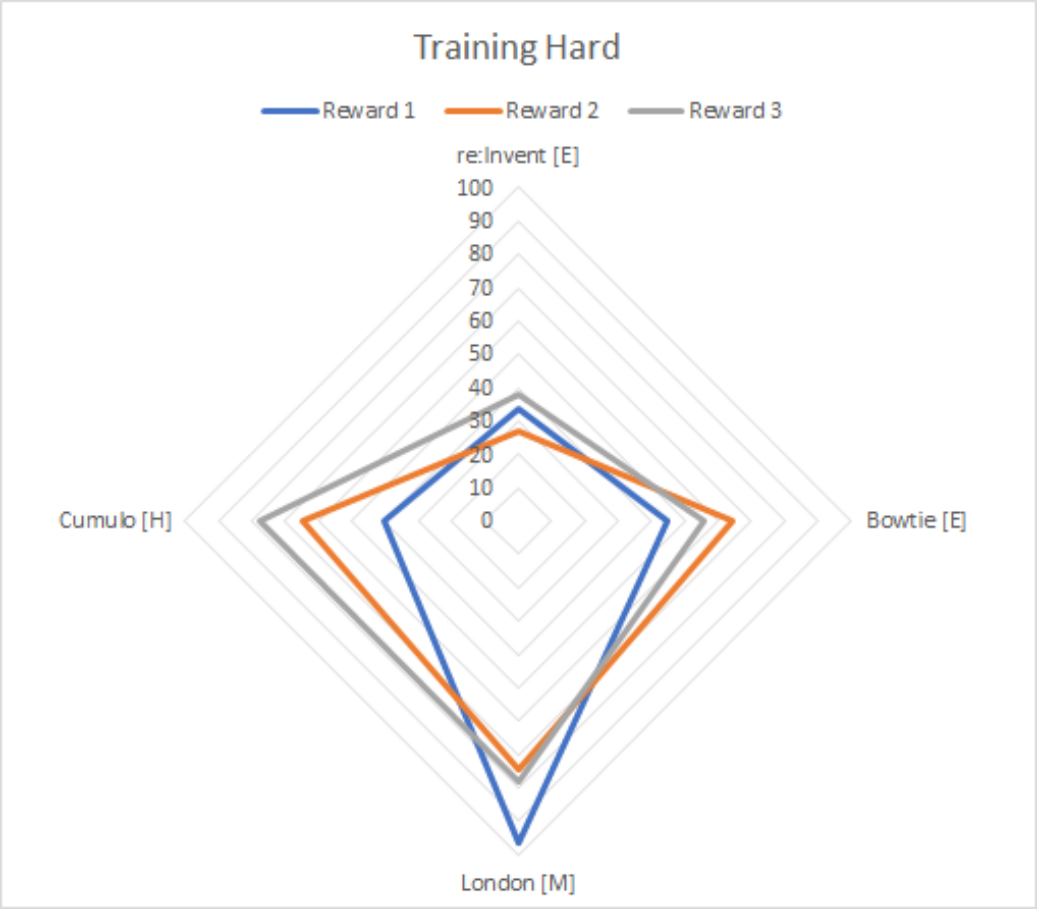

Figure 4.8: Reward functions trained in the hard track evaluation performance overview

Reward Function	CP	t Avg.	TC	Metric
1	53,7	21,79	5	14,78
2	57,5	19,3	7	23,83
3	62,3	22,9	5	16,31

Table 4.26: Evaluation Summary trained in the Hard track

In here the Figure 4.8 shows how the third reward function has a better performance when it comes to average track completion and the first one is

the worst, however when we check the Table 2.26 and we compare the metric, we can see that the second reward function is still the best one overall but on this training scenario the first reward function becomes the worst one in the overall performance metric.

Chapter 5

Conclusion

The task was to implement an algorithm for a self-propelled vehicle in the amazon web services RoboMaker virtual environment using reinforcement learning strategies that give the agent the ability to navigate a track without a definition of pre- established rules, the agent would take a single image from the frontal camera and make a decision based on the policy made based on the reward function, the lack of depth perception made the task more difficult.

We created and trained two different reward functions and using the default "Stay inside the two borders" reward function as a baseline for comparison, we evaluated them on four different tracks with different difficulties to measure the generalization capabilities of each reward function.

Both reward functions that we created were based on the "Stay inside the two borders" but added different parameters to reward the agent, the first one, called "Reward speed and progress" used the speed and the general progress, it would reward the agent higher if it goes faster and would reward higher depending on the progress in the track. The second reward function, called "Reward speed, progress, and direction," used the speed and the progress but also rewarded the agent if it would go in the same direction as the track, this was implemented to avoid the zigzagging of the agent.

For the training we used two different scenarios, the first one was to train the agent on an easy track, the second one was to train it on a hard track and then the evaluation of each model would be on four tracks including the

one where they were trained.

The evaluation showed that "Reward speed and progress" reward function had a better performance generalizing the driving policies in both scenarios, even though the "Reward speed, progress, and direction" was a more complex reward function it showed that complicated reward functions can be counterproductive when it comes to generalizing policies. When it comes to choosing the training scenario, it seems that training in the easy track allowed the agents to develop a better driving policy. One interesting fact is that from all the three reward functions, the only one that performed better after being trained on the hard track was the third reward function "Reward speed, progress and direction" this means that this reward function is better at making driving policies from harsh scenarios than easy ones.

5.1 Future Work

We only trained the agents for one hour on each scenario; this is a short time for an agent to develop a driving policy able to generalize and perform well in other tracks. Also, other reward functions could be explored; for example, having a simple reward function would be an excellent point to compare the trade-off between complex and straightforward reward functions. This project tackled the task where the agent only has to navigate through the track, no objects or other agents were sharing the track; these are two critical points for the autonomous driving task.

Bibliography

Abbeel, P., Coates, A., Quigley, M., & Ng, A. Y. (2007). An application of reinforcement learning to aerobatic helicopter flight. In *Advances in neural information processing systems* (pp. 1–8).

Abduljabbar, R., Dia, H., Liyanage, S., & Bagloee, S. A. (2019). Applications of Artificial Intelligence in Transport : An Overview. doi: 10.3390/su11010189

Chen, J., Chen, J., Zhang, R., & Hu, X. (2019). Towards Brain-inspired System: Deep Recurrent Reinforcement Learning for Simulated Self-driving Agent. Retrieved from `http://arxiv.org/abs/1903.12517`

Endo, G., Morimoto, J., Matsubara, T., Nakanishi, J., & Cheng, G. (2008). Learning cpg-based biped locomotion with a policy gradient method: Application to a humanoid robot. *The International Journal of Robotics Research*, *27*(2), 213–228.

Genc, S., Mallya, S., Balaji, B., Sun, T., Gupta, S., Khare, V., ... Calleja, E. (2019). Learning Robust Motion Planning for Large-Scale Deployment of Autonomous Driving Agents.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). *Deep learning*. MIT Press. (`http://www.deeplearningbook.org`)

Kendall, A., Hawke, J., Janz, D., Mazur, P., Reda, D., Allen, J.-M., ... Shah, A. (2018). Learning to Drive in a Day. Retrieved from `http://arxiv.org/abs/1807.00412`

Kohl, N., & Stone, P. (2004). Policy gradient reinforcement learning for fast quadrupedal locomotion. In *Ieee international conference on robotics and automation, 2004. proceedings. icra'04. 2004* (Vol. 3, pp. 2619–2624).

Liang, X., Wang, T., Yang, L., & Xing, E. (n.d.). CIRL: Controllable Imitative Reinforcement Learning for Vision-based Self-driving. (arXiv:1807.03776v1 [cs.CV]). Retrieved from

http://arxiv.org/abs/1807.03776

Mell, P., & Grance, T. (2011). The NIST-National Institute of Standars and Technology- Definition of Cloud Computing. *NIST Special Publication 800-145*, 7.

Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... others (2015). Human-level control through deep reinforcement learning. *Nature*, *518*(7540), 529.

Mnih, V., Mirza, M., Graves, A., Harley, T., Lillicrap, T. P., & Silver, D. (2016). Asynchronous Methods for Deep Reinforcement Learning. , *48*.

Mnih, V., & Silver, D. (2013). Playing Atari with Deep Reinforcement Learning. , 1–9.

NHTSA. (2019, Jun). *Automated vehicles for safety.* Retrieved from https://www.nhtsa.gov/technology-innovation/automated-vehicles-safety

Pan, X., You, Y., Wang, Z., & Lu, C. (2017). Virtual to Real Reinforcement Learning for Autonomous Driving. Retrieved from http://arxiv.org/abs/1704.03952

Rosenzweig, J., & Bartl, M. (2015). THE MAKING-OF INNOVATION, E-JOURNAL makingofinnovation A Review and Analysis of Literature on Autonomous Driving. (October), 1–57.

Sadeghi, F., Toshev, A., & Jang, E. (2017). Sim2Real View Invariant Visual Servoing by Recurrent Control.

Shalev-Shwartz, S., Shammah, S., & Shashua, A. (2016). Safe, Multi-Agent, Reinforcement Learning for Autonomous Driving. Retrieved from http://arxiv.org/abs/1610.03295

Silver, D., Schrittwieser, J., Simonyan, K., Antonoglou, I., Huang, A., Guez, A., ... Hassabis, D. (2017). Mastering the game of Go without human knowledge. *Nature*, *550*(7676), 354–359. Retrieved from http://dx.doi.org/10.1038/nature24270 http://www.ncbi.nlm.nih.gov/pubmed/29052630 doi: 10.1038/nature24270

Sutton, R. S., & Barto, A. G. (2017). Reinforcement Learning : An Introduction **** Complete Draft ****.

Tesla vehicle safety report. (2019, May). Retrieved from https://www.tesla.com/es$_MX/VehicleSafetyReport?redirect$ = *no*

Thrun, S. (2012). *50 Autonomous Driving : Context and State-of-the-* (No. March). doi: 10.1007/978-0-85729-085-4

Unlimited, D. (2004). DARPA-IPTO Arlington , VA Autonomous Off-Road Vehicle Control Using End-to-End Learning. (3).

Večerík, M., Hester, T., Scholz, J., Wang, F., Pietquin, O., Piot, B., ... Riedmiller, M. (2017). Leveraging demonstrations for deep reinforcement learning on robotics problems with sparse rewards. *arXiv preprint arXiv:1707.08817*.

Waymo. (2018). *On the road to Fully Self-driving - Waymo Safety Report* (Tech. Rep.). Retrieved from https://storage.googleapis.com/sdc-prod/v1/safety-report/waymo-safety-report- doi: 10.1016/B978-0-08-037539-7.50012-0

Xu, N., Tan, B., & Kong, B. (2018). Autonomous Driving in Reality with Reinforcement Learning and Image Translation. Retrieved from http://arxiv.org/abs/1801.05299